Balanced Binary Search Trees

- Balanced BSTs, such as red-black trees, support each of the basic dictionary operations (insert, delete, find) in $O(\log n)$ time
- Our $\Omega(n \log n)$ bound for comparison-based sorting implies that we cannot hope to insert in $o(\log n)$ time

Can we Improve on the Performance of Balanced BSTs?

- Suppose we are performing a sequence of successful find operations in an n-node BST
- If a key x is accessed frequently, we could hope to get better performance by restructuring the tree so that x is close to the root
- The rotate-to-root heuristic restructures the tree after a successful find of x by repeatedly rotating at x until x reaches the root
  - Unfortunately, this heuristic admits pathological access sequences with average find cost $\Theta(n)$

Splay Trees

- Splay trees are based on a slight variant of the rotate-to-root heuristic
- Three restructuring operations are defined: zig, zig-zag, and zig-zig
- Only the zig-zig operation distinguishes splay trees from the rotate-to-root heuristic

The "Zig" Splay Step

- Assume that node x is a child of the root
- Perform a rotation at node x

The "Zig-Zag" Splay Step

- Assume that node x has a parent p and a grandparent g, and that either x and p are both left children, or x and p are both right children
- Perform two successive rotations at node x

The "Zig-Zig" Splay Step

- Assume that node x has a parent p and a grandparent g, and that either x and p are both left children, or x and p are both right children
- Perform a rotation at p followed by a rotation at x
- This results in a different BST than we would get by performing two successive rotations at x!

The Splay Operation

- To perform a splay operation at a node x, we repeat the following until x is the root
  - If x is a child of the root, perform a zig splay step at x
  - Otherwise, let p be the parent of x, and let g be the grandparent of x
    - If exactly one of p and x is a left child, perform a zig-zag splay step at x
    - Otherwise, perform a zig-zig splay step at x

Find

- To perform a find operation in a splay tree, we begin by doing a "plain" BST find
- If a node x containing the desired key is found, we perform a splay operation at node x
- If the plain BST find terminates unsuccessfully, it does so at some node y, and we perform a splay operation at node y
- Note that the total cost is proportional to the cost of the splay operation

Insert

- Let us assume that the key x being inserted does not already belong to the tree
- We begin by doing an unsuccessful plain BST find for x, which terminates at some node y
- We add a new node with key x as the left or right child of y, as appropriate
- We perform a splay operation at the new node
- Note that the total cost is proportional to the cost of the splay operation

Join

- Before defining splay tree deletion, it is useful to define the splay tree join operation
- This operation takes two BSTs T_1 and T_2 such that every key in T_1 is less than every key in T_2, and merges them into a single BST
- We descend rightward in T_1 until we reach the node x containing the maximum key
- We perform a splay operation at x
- We make the root of T_2 the right child of x (the root of T_1)
- Note that the total cost is proportional to the cost of the splay operation

Delete

- Let us assume that the key x to be deleted is in the BST
- We perform a successful splay tree find for x, moving the associated node to the root
- We remove the root node, leaving two trees T_1 and T_2 such that every key in T_1 is less than every key in T_2
- We perform a splay tree join operation to merge trees T_1 and T_2
- Note that the actual cost is proportional to the cost of the two associated splay operations

Analyzing the Cost of a Sequence of Dictionary Operations

- To determine the asymptotic complexity of a sequence of insertion, deletion, and find operations on a splay tree, it is sufficient to bound the total cost of the splay operations
- The cost of any splay operation is $O(k+1)$ where k denotes the number of rotations, so we can focus on counting rotations

Node Weight, Size, and Rank

- We associated a positive weight $w(x)$ with each node x
  - Our main technical lemma will hold for all possible choices of the weights
  - Different weight assignments are useful for proving various properties of splay trees
- We define the size of a node x, denoted $s(x)$, as the sum of $w(y)$ over all nodes y in the subtree rooted at x
- We define the rank of a node x, denoted $r(x)$, as $\log_2 s(x)$

The Potential Function

- We define the potential $\Phi(T)$ of a splay tree T as the sum of the ranks of the nodes in T

A Useful Fact

- Fact: If x and y be two positive real numbers such that $x + y \leq 1$, then $\log_2 x + \log_2 y \leq -2$
- We can assume without loss of generality that $x + y = 1$ since $\log_2 z$ is increasing in $z > 0$
- To maximize $\log_2 x + \log_2(1 - x) = \log_2[x(1 - x)]$, we need to maximize $x(1 - x)$
- The expression $x - x^2$ is maximized at $x = 1/2$
- We have $\log_2 \frac{1}{4} = -2$

Amortized Cost of a Splay Step

- In the analysis that follows, we use the following notational conventions for any node z
  - We write $r(z)$ (resp, $s(z)$) to denote the rank (resp., size) of z before the splay step
  - We write $r'(z)$ (resp, $s'(z)$) to denote the rank (resp., size) of z after the splay step
- We will prove the following bounds
  - The amortized cost of a zig splay step on a node x is at most $3(r'(x) - r(x)) + 1$
  - The amortized cost of a zig-zag splay step on a node x is at most $3(r'(x) - r(x))$
  - The amortized cost of a zig-zig splay step on a node x is at most $3(r'(x) - r(x))$

Amortized Cost of a Zig Splay Step

- The change in potential is $r'(x) + r'(p) - r(x) - r(p)$
  - Since $r'(x) = r(p)$, this is $r'(p) - r(x)$
  - Since $r'(p) \leq r'(x)$, this is at most $r'(x) - r(x)$
- Thus the amortized cost is at most $r'(x) - r(x) + 1$
- Since $r'(x) \geq r(x)$, this is at most $3(r'(x) - r(x)) + 1$

Amortized Cost of a Zig-Zag Splay Step

- We wish to prove that $2 + \Delta\Phi$ is at most $3(r'(x) - r(x))$
- Since $r'(x) = r(g)$, this is equivalent to showing

$$2 + r'(p) + r'(g) - r(p) - r(x) \leq 3(r'(x) - r(x))$$

or

$$(r'(p) - r'(x) + r'(g) - r'(x)) - r(p) - r'(x) + 2r(x) \leq -2$$

Amortized Cost of a Zig-Zag Splay Step (cont'd)

- The technical lemma implies that

$$r'(p) - r'(x) + r'(g) - r'(x) \leq -2$$

since $s'(p) + s'(g) \leq s'(x)$, $r'(p) - r'(x) = \log_2 \frac{s'(p)}{s'(x)}$, and $r'(g) - r'(x) = \log_2 \frac{s'(g)}{s'(x)}$

- Thus it is sufficient to prove that $-r(p) - r'(x) + 2r(x) \leq 0$
- The latter inequality holds since $r(p) \geq r(x)$ and $r'(x) \geq r(x)$

Amortized Cost of a Zig-Zig Splay Step

- We wish to prove that $2 + \Delta\Phi$ is at most $3(r'(x) - r(x))$
- Since $r'(x) = r(g)$, this is equivalent to showing

$$2 + r'(p) + r'(g) - r(p) - r(x) \leq 3(r'(x) - r(x))$$

or

$$(r(x) - r'(x) + r'(g) - r'(x)) + r'(p) - r(p) - r'(x) + r(x) \leq -2$$

Amortized Cost of a Zig-Zig Splay Step (cont'd)

- The technical lemma implies that

$$r(x) - r'(x) + r'(g) - r'(x) \leq -2$$

  since $s(x) + s'(g) \leq s'(x)$, $r(x) - r'(x) = \log_2 \frac{s(x)}{s'(x)}$, and $r'(g) - r'(x) = \log_2 \frac{s'(g)}{s'(x)}$
- Thus it is sufficient to prove that $r'(p) - r(p) - r'(x) + r(x) \leq 0$
- The latter inequality holds since $r'(p) \leq r'(x)$ and $r(p) \geq r(x)$

Amortized Analysis of a Splay Operation

- We sum our upper bounds on the costs of the individual splay steps
- This sum telescopes, yielding $3(r(t) - r(x)) + 1$, where t denotes the root of the tree before the splay operation
- Next we will use this bound establish various results about the performance of splay trees

Worst-Case Cost of a Sequence of Dictionary Operations

- Suppose we perform m dictionary operations (insert, delete, find) on a splay tree that is initially empty
  - The initial potential is zero
- Set the weight of each node to 1
  - Thus the node ranks and the potential are nonnegative
  - Thus the sum of the amortized costs is an upper bound on the sum of the total number of rotations
  - For an n-node tree, the rank of the root is $\log_2 n$, and the amortized cost of a splay operation is at most $3 \log_2 n + 1$
- Thus, the worst-case asymptotic complexity of splay trees matches that of balanced BSTs

The Balance Theorem

- Theorem: The cost of performing m find operations on an arbitrary n-node initial BST is $O((m+n)\log n)$
- As in the previous analysis (with node weights again set to 1), we the amortized cost of each operation is $O(\log n)$
- However, the initial potential can be nonzero
  - It is at most $n \log_2 n$ since each node has rank at most $\log_2 n$
  - It can be as high as $\sum_{1 \le i \le n} \log_2 i \sim n \log_2 n$
- The $n \log_2 n$ term in the theorem statement upper bounds the drop in potential (over the entire sequence)

The Static Optimality Theorem

- Consider a sequence of m find operations performed on an arbitrary initial n-node BST
- Assume that key x_i is accessed $q_i \geq 1$ times
- Theorem: The total cost is

$$O\left(m + \sum_{1 \leq i \leq n} q_i \log_2 \frac{m}{q_i}\right)$$

- Remark: It can be shown that the optimal static BST pays $\Omega(\log_2(m/q_i))$ for each access to item i, thereby implying a total cost of

$$\Omega\left(m + \sum_{1 \leq i \leq n} q_i \log_2 \frac{m}{q_i}\right)$$

Proof of the Static Optimality Theorem

- Assign weight q_i/m to key x_i
  - Thus the rank of the root is zero, and $0 \geq r(x_i) \geq \log_2 \frac{q_i}{m} = -\log_2 \frac{m}{q_i}$
- The amortized cost of each access to x_i is at most

$$3\left(0 + \log_2 \frac{m}{q_i}\right) + 1 = O\left(1 + \log_2 \frac{m}{q_i}\right)$$

- Thus the total amortized cost meets the O-bound stated in the theorem
- It remains to bound the drop in potential, which is at most $\sum_{1 \leq i \leq n} \log_2 \frac{m}{q_i}$, which corresponds to a subset of the terms in the O-bound of the theorem

The Working-Set Theorem

- Consider a sequence of m successful finds performed on an arbitrary initial n-node BST
- Let t_i denote the number of distinct key accessed between the ith access and the previous access to the same item
  - If the ith access is the first access to a key, we define t_i as the number of distinct keys accessed prior to the ith access
- Theorem: The total cost is

$$O\left(m + n \log n + \sum_{1 \le i \le m} \log(t_i + 1)\right)$$

The Sequential Access Theorem

- Let T be an n-node BST with keys $x_1 < \ldots < x_n$
- Suppose that we do a find for x_1, then x_2, and so on, up to x_n
- Theorem: The total cost is $O(n)$

The BST Model

- We augment our usual notion of a BST with a "finger" that points to some node
- The following elementary operations are allowed
  - Move the finger to an adjacent node (left child, right child, or parent)
  - Rotate the node pointed at by the finger with its parent
  - Return the node pointed at by the finger
- Note that the find operation of a splay tree conforms to this model

Dynamic Optimality Conjecture

- Let T be an n-node BST with keys $x_1 < \ldots < x_n$
- Let Q be a sequence of successful find queries on T
- Let $f(T, Q)$ be the minimum number of elementary operations used by any algorithm in the BST model that correctly processes Q starting from BST T
- Let $g(T, Q)$ denote the number of elementary operations used by a splay tree to process Q starting from BST T
- Conjecture: $g(T, Q) = O(f(t, Q) + n)$